XML Schema Regular Expressions - Basics

Introduction

The XML Schema language has its own regular expression syntax and specialized notation. It is very similar to the syntax used by Perl, but not sufficiently similar that you can plug in even a simple Perl regular expression and expect it to work as it would in a Perl script. The full syntax available for use in XML Schema 1.1 documents is defined in Appendix G of the W3C Standard: https://www.w3.org/TR/xmlschema11-2/

Data dictionary designers might encounter XML Schema regular expressions if they make use of the <pattern> option in defining their local attributes. Patterns are used by PDS to define some specific data types in the pds: core namespace, so label designers and creators may also encounter them in schema files while investigating validation failures.

What follows is *not* an exhaustive guide to regular expressions in XML Schema. Rather, it is a quick review of the features most likely to be encountered in the pds: schema or to be useful to dictionary writers. It also assumes you already have familiarity with regular expressions in some other context. More complete descriptions are readily available on the web, should you wish to delve deeper.

This is also aimed at PDS data preparers and users, so it assumes that the patterns are being developed for use in constraining and validating PDS4 label attributes.

Before You Even Start

Here are a couple things you should be aware of to start:

- *All regular expressions in XML Schema are anchored to both the beginning and ending of the string being compared.* So all patterns must match the first character and all characters through to the end of the input string in order to be considered successful. The standard anchor characters (^ and *$*) will cause failures or syntax errors if you try to use them as anchors.
- When defining patterns, *do not* enclose them in delimiters. So if you are trying to define a *<pattern>* facet for a data type in your local dictionary, do this:
  `<pattern>[A-Z][a-z]+</pattern>`
  not this:
  `<pattern>/[A-Z][a-z]+/</pattern>`

The Usual

These points should be familiar from using regular expressions in other contexts:

- Single characters match themselves literally unless they are special characters.
- Character classes are defined using '[]'. The hyphen ('-') can be used to indicate a range and the carat ('^') negates the class if it is the first character in the class.
- The wildcard character '.' matches any single character.
- The quantifier character '?' means "zero or one time"; '*' means "zero or more times"; '+' means "one or more times".
- The quantifier expression "{*n*}" means "exactly *n* times; "{*n*,}" means "*n* or more times"; "{*n*,*m*}" means "at least *n* but not more than *m* times".
- Parentheses ('()') can be used to group.

- A vertical bar indicates alternation between two patterns. The pattern `abc|def` matches either *abc* or *def*, but not *abdef*. The pattern `ab(c|d)ef` matches *abdef*, but not *abc* or *def*.
- The escape character is '\'.
- A space character is taken as literal - that is, it matches exactly one space character

In XML Schema

The following conventions are not necessarily specific to XML Schema regular expression syntax, but they are not quite as universal as the preceding conventions.

Character Entities

The XML Schema character entities *&lt;*, *&gt;*, *&amp;*, *&apos;*, and *&quot;* can be used to match the characters they represent (<, >, &, *'*, and *"*, respectively). So if you need to check for a value that must contain an ampersand, you can use a pattern like this:

```
<pattern>.*&amp;.*</pattern>
```

This will match either the character '&', or the entity reference '&amp;' - whichever appears in your label. (The `.*` at the beginning and end of the pattern ignores everything that might be around the desired ampersand.)

Character References

XML character references will also be matched as the character they correspond to. Character references begin with "&#" followed by a decimal number and a semicolon (';'), or "&#x" followed by a hexadecimal number and a semicolon. The number part corresponds to the Unicode code point for the character. So the character reference for a blank space, for example, is "&#x20;". Either an explicit blank or the character reference will be matched, so these two patterns are equivalent:

```
<pattern>Hello World</pattern>
<pattern>Hello&#x20;World</pattern>
```

and both will match either "Hello World" or "Hello&#x20;World" in your XML label.

Escape Sequences

These sequences match one character of the type indicated:

\n
matches a linefeed character
\r
matches a carriage-return character
\t
matches a horizontal tab character
\d
matches any decimal digit (*D* is the negation)
\s
matches any whitespace character (*S* is the negation)

All types of quantifiers can be used with these escape sequences - so "*n*+" matches one or more newline characters in a row.

There are a few more of these escapes defined by the XML Schema specification to cover cases like characters that might be part of a valid XML name. Check the XML Schema 1.1 standard Appendix G if you are looking for something along these lines.

Block Escapes

Block escapes provide a method for matching against contiguous blocks of Unicode characters. As with the preceding escapes, a block escape matches exactly one character unless is it followed by a quantifier.

For PDS purposes, there is really only one block escape to be concerned with - the *BasicLatin* block, which spans the ASCII code page. If you want to constrain your values to use only ASCII characters, use a block escape that looks like this:

`<pattern>\p{IsBasicLatin}+<\pattern>`

Gotchas

- *If you are a PDS4 dictionary author* and you are including patterns in your input file, note that *LDDTool* **cannot** detect syntax errors in your patterns. So if you have pattern facets defined in your dictionary input file, *always* validate the output XML Schema file (by opening it in a validating editor, for example) to make sure there are no syntax errors in your patterns. An unnoticed pattern error in the .xsd file will generally cause validation failures with messages complaining about not being able to find or read the schema file.
- If you don't normally work in a Unicode environment, it is important to remind yourself frequently that XML *is* very firmly in the Unicode world. It is often better to be specific about characters and digits rather than relying on generic escape sequences. The `\d` escape, for example, will also happily match non-European digits. If you really mean "0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and nothing else", it is better to use the character class `[0-9]` rather than `\d`. Similarly for `\w`, which selects "word characters" by their Unicode properties, not their code page.
- The "+" character is a metacharacter (unless it is inside "[ ]"). If you want to match it explicitly (in checking for a signed number, for example) you will have to escape it. So to require a sign followed by four digits, use this:
  `<pattern>(\+|-)[0-9]{4}</pattern>`
  not this:

  `<pattern>(+|-)[0-9]{4}</pattern>`
  The latter will produce a syntax error in the schema file.
- Remember, "<" and ">" are special characters to XML. Use the "&lt;" and "&gt;" character entities in your patterns if you need to match them.
- Also remember that all patterns are anchored at both ends. Use `.*` to skip zero or more arbitrary characters at the beginning or end of the pattern if you need to match only a substring. This pattern, for example, will check for the word "Copyright" at the beginning of a value an ignore what follows it:
  `<pattern>Copyright.*</pattern>`

Bells and Whistles

- You are unlikely to need it, but the XML Schema 1.1 regular expression syntax allows you to define a character class as one set of characters minus another set. So if you took an unnatural dislike to the number '4', you could disallow it from your numeric value with a pattern like this:
  `<pattern>[0-9-[4]]+<\pattern>`
  The inner set of brackets ("[ ]") indicates the class of characters to be excluded. *Do not* bracket the other class characters - that is syntactically incorrect. You can think of the above class as "The numbers 0-9, minus the members of [4]".

Along the same lines, this pattern would select only letters that appeared on old-style telephone dials:

```
<pattern>[A-Z-[QZ]]</pattern>
```

- If you specify multiple patterns when defining a data type, then a value of that type only has to match one of those patterns to be considered valid, not all of them. If the value doesn't match any of the patterns, it is flagged as invalid.
- The XML Schema standard defines some escape categories that match characters based on their defined Unicode characteristics, like "Is an uppercase letter", or "Is a numeric digit". These are not generally useful in a PDS context because PDS requires English as the standard language and European digits for labels and documentation - so values with non-English letters and other numeric digits are prohibited by the larger context. But if you have a text field that allows or even requires non-ASCII characters, declare it as a UTF-8 data type rather than an ASCII one, and of course then you might want to expand your escape usage accordingly.